```
Start coding or generate with AI.
```

```
corpus = """
artificial intelligence is transforming modern society.
it is used in healthcare finance education and transportation.
machine learning allows systems to improve automatically with experience.
data plays a critical role in training intelligent systems.
large datasets help models learn complex patterns.
deep learning uses multi layer neural networks.
neural networks are inspired by biological neurons.
each neuron processes input and produces an output.
training a neural network requires optimization techniques.
gradient descent minimizes the loss function.

natural language processing helps computers understand human language.
text generation is a key task in nlp.
language models predict the next word or character.
recurrent neural networks handle sequential data.
lstm and gru models address long term dependency problems.
however rnn based models are slow for long sequences.

transformer models changed the field of nlp.
they rely on self attention mechanisms.
attention allows the model to focus on relevant context.
transformers process data in parallel.
this makes training faster and more efficient.
modern language models are based on transformers.
"""



import random
from collections import defaultdict

tokens = corpus.lower().replace(".", "").split()

def build_bigram(tokens):
    bigram = defaultdict(list)
    for i in range(len(tokens) - 1):
        bigram[tokens[i]].append(tokens[i + 1])
    return bigram

def build_trigram(tokens):
    trigram = defaultdict(list)
    for i in range(len(tokens) - 2):
        key = (tokens[i], tokens[i + 1])
        trigram[key].append(tokens[i + 2])
    return trigram

def generate_bigram_text(model, start_word, length=20):
    word = start_word
    output = [word]

    for _ in range(length):
        if word not in model:
            break
        word = random.choice(model[word])
        output.append(word)

    return " ".join(output)

def generate_trigram_text(model, start_words, length=20):
    w1, w2 = start_words
    output = [w1, w2]

    for _ in range(length):
        key = (w1, w2)
        if key not in model:
            break
        w3 = random.choice(model[key])
        output.append(w3)
        w1, w2 = w2, w3

    return " ".join(output)
```

```
bigram_model = build_bigram(tokens)
trigram_model = build_trigram(tokens)

print("Bigram Output:\n")
print(generate_bigram_text(bigram_model, "artificial", 25))

print("\nTrigram Output:\n")
print(generate_trigram_text(trigram_model, ("artificial", "intelligence"), 25))
```

```
Bigram Output:

artificial intelligence is transforming modern language models are slow for long sequences transformer models are inspired by bi

Trigram Output:

artificial intelligence is transforming modern society it is used in healthcare finance education and transportation machine lea
```

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np

corpus = """
artificial intelligence is transforming modern society.
it is used in healthcare finance education and transportation.
machine learning allows systems to improve automatically with experience.
data plays a critical role in training intelligent systems.
large datasets help models learn complex patterns.
deep learning uses multi layer neural networks.
neural networks are inspired by biological neurons.
each neuron processes input and produces an output.
training a neural network requires optimization techniques.
gradient descent minimizes the loss function.
"""

# Clean text
corpus = corpus.lower().replace(".", "")
words = corpus.split()

# Vocabulary
vocab = sorted(set(words))
word_to_idx = {word: i for i, word in enumerate(vocab)}
idx_to_word = {i: word for word, i in word_to_idx.items()}

vocab_size = len(vocab)

sequence_length = 3
X = []
y = []

for i in range(len(words) - sequence_length):
    seq = words[i:i+sequence_length]
    target = words[i+sequence_length]

    X.append([word_to_idx[word] for word in seq])
    y.append(word_to_idx[target])

X = torch.tensor(X)
y = torch.tensor(y)


class RNNModel(nn.Module):
    def __init__(self, vocab_size, embed_size=32, hidden_size=64):
        super(RNNModel, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.rnn = nn.RNN(embed_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, vocab_size)

    def forward(self, x):
        x = self.embedding(x)
        out, _ = self.rnn(x)
        out = self.fc(out[:, -1, :])  # Use last output
        return out
```

```
model = RNNModel(vocab_size)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)


epochs = 300

for epoch in range(epochs):
    optimizer.zero_grad()
    outputs = model(X)
    loss = criterion(outputs, y)
    loss.backward()
    optimizer.step()

    if (epoch+1) % 50 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")


def generate_text(seed_text, num_words=20):
    model.eval()

    words_list = seed_text.lower().split()

    for _ in range(num_words):
        seq = words_list[-sequence_length:]
        seq_idx = torch.tensor([[word_to_idx[word] for word in seq]])

        with torch.no_grad():
            output = model(seq_idx)
            predicted_idx = torch.argmax(output).item()

        predicted_word = idx_to_word[predicted_idx]
        words_list.append(predicted_word)

    return " ".join(words_list)
print("\n--- RNN Generated Text ---\n")
print(generate_text("artificial intelligence is", 20))
```

```
Epoch [50/300], Loss: 0.0017
Epoch [100/300], Loss: 0.0010
Epoch [150/300], Loss: 0.0008
Epoch [200/300], Loss: 0.0006
Epoch [250/300], Loss: 0.0005
Epoch [300/300], Loss: 0.0004

--- RNN Generated Text ---

artificial intelligence is used in healthcare finance education and transportation machine learning allows systems to improve au
```

```
import torch
import torch.nn as nn
import torch.optim as optim
import math



corpus = """
artificial intelligence is transforming modern society
machine learning allows systems to improve automatically with experience
data plays a critical role in training intelligent systems
transformer models changed the field of nlp
attention allows the model to focus on relevant context
modern language models are based on transformers
"""

words = corpus.lower().split()

vocab = sorted(set(words))
word_to_idx = {w: i for i, w in enumerate(vocab)}
idx_to_word = {i: w for w, i in word_to_idx.items()}

vocab_size = len(vocab)

data = [word_to_idx[w] for w in words]
data = torch.tensor(data)
```

```
X = data[:-1]
y = data[1:]

X = X.unsqueeze(0)  # batch dimension
y = y.unsqueeze(0)



class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=500):
        super().__init__()

        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len).unsqueeze(1)
        div_term = torch.exp(
            torch.arange(0, d_model, 2) * (-math.log(10000.0) / d_model)
        )

        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)

        pe = pe.unsqueeze(0)
        self.register_buffer("pe", pe)

    def forward(self, x):
        x = x + self.pe[:, :x.size(1)]
        return x



class TransformerModel(nn.Module):
    def __init__(self, vocab_size, d_model=128, nhead=4, num_layers=2):
        super().__init__()

        self.embedding = nn.Embedding(vocab_size, d_model)
        self.pos_encoder = PositionalEncoding(d_model)

        encoder_layer = nn.TransformerEncoderLayer(
            d_model=d_model,
            nhead=nhead,
            dim_feedforward=256,
            batch_first=True
        )

        self.transformer = nn.TransformerEncoder(
            encoder_layer,
            num_layers=num_layers
        )

        self.fc = nn.Linear(d_model, vocab_size)

    def forward(self, x):
        x = self.embedding(x)
        x = self.pos_encoder(x)

        mask = nn.Transformer.generate_square_subsequent_mask(x.size(1)).to(x.device)

        x = self.transformer(x, mask)
        out = self.fc(x)
        return out



model = TransformerModel(vocab_size)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.005)

epochs = 200



for epoch in range(epochs):
    optimizer.zero_grad()

    output = model(X)
```

```
        loss = criterion(
            output.view(-1, vocab_size),
            y.view(-1)
        )

        loss.backward()
        optimizer.step()

        if (epoch+1) % 50 == 0:
            print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")



def generate_text(seed_text, num_words=15):
    model.eval()

    words_list = seed_text.lower().split()

    for _ in range(num_words):
        input_seq = torch.tensor(
            [[word_to_idx[w] for w in words_list]]
        )

        with torch.no_grad():
            output = model(input_seq)

        next_word_logits = output[0, -1]
        predicted_idx = torch.argmax(next_word_logits).item()

        predicted_word = idx_to_word[predicted_idx]
        words_list.append(predicted_word)

    return " ".join(words_list)

print("\nGenerated Text:\n")
print(generate_text("artificial intelligence", 15))
```

```
Epoch 50, Loss: 0.0108
Epoch 100, Loss: 0.0140
Epoch 150, Loss: 0.1277
Epoch 200, Loss: 0.2420

Generated Text:

artificial intelligence is transforming modern society machine learning allows systems to improve automatically data plays a cri
```